Robust Control Toolbox™ Release Notes

How to Contact MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support
suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Robust Control Toolbox™ Release Notes

Contents

R2014a

R2013b

R2013a

R2012b

R2012a

R2011b

R2011a

R2010b

R2010a

R2009b

R2009a

R2008b

R2008a

R2007b

R2007a

R2006b

R2006a

R14SP3

R2014a

Version: 5.1

New Features: Yes

Bug Fixes: Yes

Control System Tuner app for automated tuning of control systems

The new Control System Tuner lets you interactively tune SISO or MIMO control systems modeled in MATLAB® or Simulink®. Control System Tuner tunes the control system parameters to meet design requirements you specify, such as reference tracking, disturbance rejection, stability margins, loops shapes, and sensitivity. You can examine multiple system responses in both the time and frequency domains to evaluate performance of the tuned control system.

If you have Simulink Control Design™ software, you can tune a control system represented by a Simulink model. Control System Tuner can tune most blocks used to create a control system in Simulink. These blocks include `Gain`, `PID Controller`, `Transfer Fcn`, `State-Space`, `Zero-Pole`, `Discrete Filter`, and the `LTI System` block. Any controller architecture created using these blocks can be tuned. To access Control System Tuner for tuning a Simulink model, select **Analysis > Control Design > Control System Tuner**.

Control System Tuner can also tune a control system represented by a tunable `genss` model. Any control architecture constructed with Control Design Blocks such as `ltiblock.pid`, `ltiblock.tf`, or `realp` blocks can be tuned. To open Control System Tuner for tuning a control system modeled in MATLAB, use the `controlSystemTuner` command.

For more information about using Control System Tuner, see:

- "Automated Tuning Basics"
- "Tuning with Control System Tuner"

Step response and LQG requirements for control system tuning with `systune` and `looptune` commands

New `TuningGoal` requirement objects allow you to specify tuning objectives for automated tuning of control systems with `systune` and `looptune`.

- `TuningGoal.StepResp` — Requires that the step response between specified locations in the control system match the step response of a specified reference system. For details about this requirement, see the `TuningGoal.StepResp` reference page.

- `TuningGoal.LQG` — Specifies a linear-quadratic-gaussian (LQG) goal for control system tuning. This requirement lets you quantify control performance as an LQG cost. For details about this requirement, see the `TuningGoal.LQG` reference page.

Improvements to `TuningGoal` requirements for control system tuning

Compatibility Considerations: Yes

This release introduces a variety of improvements to `TuningGoal` requirement objects for automated tuning of fixed-structure control systems with `systune` and `looptune`.

Tuning Goals for constraining dynamics impose implicit stability constraints

`TuningGoal.StableController` and `TuningGoal.Poles` now impose implicit stability constraints on controller or system dynamics. This allows you to require poles of the controller or the closed-loop control system to be stable, without necessarily limiting the minimum decay or maximum frequency of those poles. Previously, you had to specify finite values for minimum decay and maximum frequency when using these tuning goals.

Compatibility Considerations

The default values of the `MinDecay` and `MaxFrequency` properties of these requirements have changed. If you have scripts that use `TuningGoal.StableController` or `TuningGoal.Poles` requirements with default values, update those scripts to explicitly set the finite values you want.

Property	Previous Default Value	New Default Value
`TuningGoal.Poles.MinDecay` `TuningGoal.StableController.MinDecay`	1e-6	0
`TuningGoal.Poles.MaxFrequency` `TuningGoal.StableController.MaxFrequency`	1e6	Inf
`TuningGoal.Poles.MinDamping`	1e-6	0

Option to limit dynamics constraint to poles in a particular feedback loop

A new syntax for creating the `TuningGoal.Poles` requirement allows you to constrain only the poles of the sensitivity function measured at a specified location. Use this syntax to narrow the scope of the requirement to a particular feedback loop.

For example, suppose you have a cascaded-loop control system in which the inner and outer loops contain loop-opening locations `'InnerLoop'` and `'OuterLoop'`, respectively. The following command uses the new syntax to constrain the poles of the inner loop sensitivity function:

```
Req = TuningGoal.Poles('InnerLoop');
Req.MinDamping = 0.5;
Req.Openings = 'OuterLoop';
```

`Req` imposes a minimum damping on the poles of the inner loop sensitivity function measured with the outer loop open. The dynamics of blocks that do not participate to the inner loop are ignored.

For more information about using this constraint, see the `TuningGoal.Poles` reference page.

`TuningGoal.Tracking` allows specification of peak error

A new syntax for creating the `TuningGoal.Tracking` requirement allows you to specify a maximum tracking error for a particular input-output pair in terms of a response time, a relative DC error, and a peak relative error across all frequencies. These parameters are converted to the following expression for the maximum tracking error:

$$\text{MaxError} = \frac{(\text{PeakError})s + \omega_c(\text{DCError})}{s + \omega_c}.$$

For more information about how to specify tracking error requirements, see the `TuningGoal.Tracking` reference page.

Specification of signal scaling in MIMO closed-loop Tuning Goals

New properties in several closed-loop Tuning Goals allow you to specify the relative amplitudes of multiple input and output signals in the loops constrained by the requirements. Use these properties to reduce cross-coupling in tuned systems when the choice of units results in a mix of small and large signals.

- `TuningGoal.Tracking` and `TuningGoal.Overshoot` now have an `InputScaling` property. This information is used to scale the off-diagonal terms in the transfer function from reference to tracking error. This scaling ensures that cross-couplings are measured relative to the amplitude of each reference signal.
- `TuningGoal.Gain` and `TuningGoal.Variance` now have `InputScaling` and `OutputScaling` properties. The values you set for these properties are used to scale the closed-loop transfer function $T(s)$ on which you impose the tuning requirement. The requirement is evaluated for the scaled transfer function $D_o^{-1}T(s)D_i$. D_o and D_i are diagonal matrices formed from the `OutputScaling` and `InputScaling` property, respectively.

For more information on how to interpret and use these properties, see the reference pages for the Tuning Goals.

Option to remove stability constraint from loop-shape and gain-limiting Tuning Goals

The new `Stabilize` property of loop-shaping and gain-limiting Tuning Goals allows you turn off the implicit closed-loop stability constraint. If stability for the specified loop is not required or cannot be achieved, set `Stabilize` to `false` to relax the stability constraint.

This property is available for the following Tuning Goals:

- `TuningGoal.LoopShape`
- `TuningGoal.Gain`, `TuningGoal.WeightedGain`
- `TuningGoal.MinLoopGain`,`TuningGoal.MaxLoopGain`

For more information on how to use the `Stabilize` property, see the reference pages for the Tuning Goals.

`ScalingOrder` property added to `TuningGoal.Margins`

The `TuningGoal.Margins` tuning goal has a new property, `ScalingOrder`. This property controls the number of states in the diagonal scalings involved in computing MIMO stability margins. Increasing the order may improve results at the expense of increased computations.

Previously, this scaling order was set as a tuning option in `systuneOptions`.

Compatibility Considerations

If you have scripts that use the `ScalingOrder` option of `systuneOptions`, set the `ScalingOrder` property of `TuningGoal.Margins` instead.

Improved control system tuning of Simulink models with `systune` or `looptune` functions using `slTuner` interface (with Simulink Control Design)

Compatibility Considerations: Yes

Use the new `slTuner` interface for tuning control systems in Simulink models. This interface replaces `slTunable`. The `slTuner` interface allows you to:

- Tune model blocks and subsystems to meet tuning goals using the `systune` and `looptune` functions.

- Perform robust tuning of a controller against a set of plant models using `systune`. You can configure an `slTuner` interface to vary model parameter values and operating points. When you call `systune` for the interface, the software returns a control system that satisfies the tuning goals for all the specified model variations.
- Validate the controller design by examining the transfer function for relevant I/O sets using the `getIOTransfer`, `getLoopTransfer`, `getSensitivity`, and `getCompSensitivity` functions.

`slTuner`, similar in design to `slLinearizer`, simplifies I/O management in the controller tuning and validation workflow. You specify signals of interest as *analysis points*. You can use these analysis points to configure design requirements and specify linearization inputs/outputs when you extract transfer functions.

For more information on command-line tuning of Simulink models with `slTuner`, see:

- "Programmatic Control System Tuning"

  "Loop-Shaping Design"

Compatibility Considerations

The `slTunable` interface will continue to work for backward compatibility. However, only the `slTuner` interface will be supported and enhanced in future releases. Therefore, adoption of the `slTuner` interface is strongly recommended.

For documentation of the `slTunable` interface, see `slTunable` in the R2013b documentation.

R2013b

Version: 5.0

New Features: Yes

Bug Fixes: Yes

Automatic tuning of gain-scheduled control systems with `systune` and `looptune` commands

You can now use `systune` and `looptune` to automatically tune control systems in which plant dynamics change with operating conditions or time. In such gain-scheduled control systems, the controller gains vary as a function of one or more scheduling variables. You parameterize the dependency of controller gains on the scheduling variables. The software automatically tunes the coefficients of that parametrization so that the control system meets the tuning requirements you specify over the entire range of plant operating conditions. The new `gainsurf` command helps you parametrize your controller gains as functions of scheduling variables.

Several new examples illustrating the workflow for gain-scheduled tuning, including:

- Tuning of Gain-Scheduled Three-Loop Autopilot
- Gain Scheduled Control Of a Chemical Reactor

For additional information about tuning gain-scheduled controllers, see Gain-Scheduled Controllers.

Automatic tuning of discrete-time control systems with `systune` and `looptune` commands

You can now use `systune` and `looptune` for automatic tuning of discrete-time control systems. This capability includes both:

- Control systems represented by discrete-time generalized LTI models (`genss` models with `Ts` property not equal to zero).
- Control systems represented by an `slTunable` interface to a Simulink mode. Set the `Ts` property of the `slTunable` interface to the sampling time at which you want to linearize the model.

To tune a discrete-time control system, use the same procedure and command syntax and you use to tune a continuous-time control system. For examples of discrete-time tuning, see:

- Digital Control of Power Stage Voltage
- MIMO Control of Diesel Engine

Sensitivity, overshoot, minimum and maximum loop gain requirements for control system tuning with `looptune` and `systune`

New `TuningGoal` requirement objects allow you to specify a variety of tuning objectives for automated tuning of fixed-structure control systems with `systune` and `looptune`. New tuning requirements include:

- `TuningGoal.Sensitivity` — Constraint on sensitivity to disturbance
- `TuningGoal.Overshoot` — Constraint on overshoot in step response
- `TuningGoal.MinLoopGain` — Minimum loop gain constraint
- `TuningGoal.MaxLoopGain` — Maximum loop gain constraint

Additionally, `TuningGoal.LoopShape` has two new syntaxes. These syntaxes allow you to specify a target crossover frequency or range of crossover frequencies for an open-loop response in your control system.

For more information about these `TuningGoal` requirement objects see the reference pages for each requirement object, and:

- Using Design Requirement Objects
- Specifying Design Requirements for systune
- Performance and Robustness Specifications for looptune

`looptuneSetup` command for switching from `looptune` to `systune` to use additional `systune` functionality

The new `looptuneSetup` command provides a bridge between the tuning commands `looptune` and `systune`. `looptuneSetup` takes the argument list for `looptune` and constructs an equivalent argument list for `systune`. The `looptuneSetup` command is valid for systems represented in either MATLAB or Simulink.

You can use this command to switch from `looptune` to `systune` to take advantage of the additional flexibility and functionality of `systune`. For example, `looptune` requires that you tune all channels of a MIMO feedback loop to the same target bandwidth. Converting to `systune` allows you to specify different crossover frequencies and loop shapes for each loop in your control system. Also, `looptune` treats all tuning requirements as soft requirements, optimizing them but not requiring that any constraint be exactly met. Converting to `systune` allows you to enforce some tuning requirements as hard constraints, while treating others as soft requirements.

You can also use `looptuneSetup` to probe into the tuning requirements that `looptune` implicitly imposes. When you use `looptune`, you specify a target loop bandwidth and stability margins. `looptune` expresses these as hard and soft tuning constraints, specified as `TuningGoal` objects. You can use `looptuneSetup` to examine these constraints. After examining the constraints, you can then alter them and pass them to `systune` for further tuning.

For more information, see the following reference pages:

- `looptuneSetup`
- `slTunable.looptuneSetup`

`hinfnorm` command for computing H$_\infty$ norm

The new `hinfnorm` command computes the H_∞ norm of SISO or MIMO systems. For SISO systems, the H_∞ norm is defined as the largest value of the frequency response magnitude. For MIMO systems, H_∞ norm is the largest singular value across frequencies.

For more information, see the `hinfnorm` reference page.

Some properties of `TuningGoal` requirements renamed

Compatibility Considerations: Yes

The following properties of `TuningGoal` requirement objects are renamed to better reflect their purpose and uses:

Object	Previous Property Name	New Property Name
TuningGoal.LoopShape	LoopTransfer	Location
TuningGoal.Margins	LoopTransfer	Location
TuningGoal.Tracking	ReferenceInput	Input
TuningGoal.Tracking	TrackingOutput	Output

Compatibility Considerations

If you have scripts or functions that use any of these properties, consider updating your code to use the new property names instead. Using the previous property names does not generate an error in this release, but the names might be removed in a future release.

Power iteration method option for structured singular value computation with mussv

Compatibility Considerations: Yes

A new 'p' option to the mussv command allows you to specify a power iteration method for computing the lower bound on structured singular values (μ values). This method is recommended for cases of complex uncertainty. When at least one of the uncertain blocks specified in the block diagonal matrix structure is complex, mussv now uses the power iteration method by default.

For pure real uncertainty, mussv uses a gain-based lower bound algorithm by default.

For more information, see the mussv reference page.

Compatibility Considerations

Previously, mussv used a gain-based lower bound algorithm for both pure real and mixed uncertainty. Therefore, you might now obtain different results for the lower bounds with mixed uncertainty.

Option to specify feedback sign for stability margin calculation with `ncfmargin`

Compatibility Considerations: Yes

The `ncfmargin` command includes a new input argument that lets you specify the sign of the feedback interconnection assumed for the margin calculation. Use the syntax `[marg,freq] = ncfmargin(P,C,sign)` or `[marg,freq] = ncfmargin(P,C,sign,tol)` to specify a negative or positive feedback interconnection. For more information, see the `ncfmargin` reference page.

Compatibility Considerations

Previously, the relative accuracy `tol` was the third input argument to `ncfmargin`. If you have scripts or functions that use the syntax `[marg,freq] = ncfmargin(P,C,tol)`, update them to use `[marg,freq] = ncfmargin(P,C,-1,tol)` instead.

R2013a

Version: 4.3

New Features: Yes

Bug Fixes: Yes

Minimum damping requirement for closed-loop poles in `TuningGoal.Poles` object

You can now specify the minimum damping ratio of closed-loop poles for automated tuning of fixed-structure control systems with `systune` or `looptune`. To do so, create a `TuningGoal.Poles` object and set its `MinDamping` property to the minimum damping ratio you want to specify. Additionally, you can now use the `Focus` property to limit enforcement of the `TuningGoal.Poles` requirements to poles within a specified frequency range.

For more information about the `TuningGoal.Poles` requirement, see the `TuningGoal.Poles` reference page. For more information about using requirement objects to tune control systems, see Using Design Requirement Objects.

`TuningGoal.Rejection` object for specifying disturbance rejection requirement

You can now specify a disturbance rejection requirement for automated tuning of fixed-structure control systems with `systune` or `looptune`. The new `TuningGoal.Rejection` object allows you to specify a frequency-dependent attenuation factor for a disturbance injected at a specified location in the control system.

For more information about the `TuningGoal.Rejection` requirement, see the `TuningGoal.Rejection` reference page. For an example, see PID Tuning for Setpoint Tracking vs. Disturbance Rejection.

For more information about using requirement objects to tune control systems generally, see Using Design Requirement Objects.

`looptune` returns detailed results from multiple random starts

Compatibility Considerations: Yes

The `info` output of `looptune` now includes detailed results from each optimization run. When you use the `RandomStart` option of `looptuneOptions` to perform multiple optimization runs, the field `info.Runs` of the `info` output now contains a `struct` array. Each entry in the `struct` array includes results from the corresponding optimization run such as minimum constraint values and tuned block values. You can optionally use this information to analyze independent optimization results.

See the `looptune` reference page for more information.

Compatibility Considerations

The `Extra` field of `info` is now renamed to `Runs`. If you use `info.Extra` in a script, update your code to use `info.Runs` instead.

Additional automated tuning examples

New examples in this release include:

- Multi-Loop Control of a Helicopter
- Fault-Tolerant Control of a Passenger Jet
- Multi-Loop PID Control of a Robot Arm

R2012b

Version: 4.2

New Features: Yes

Bug Fixes: Yes

systune command for multiobjective tuning with soft and hard constraints

The new systune command allows automated tuning of fixed-structure control systems to high-level tuning objectives.

To use systune, you specify tuning objectives such as reference tracking, disturbance rejection, or stability margins. You can specify both soft requirements (objectives) and hard requirements (constraints). systune automatically tunes the parameters of your control system to meet the requirements.

You can use systune to tune control systems modeled in either MATLAB or Simulink.

For more information, see:

- Tuning Control Systems with SYSTUNE
- Tuning Control Systems in Simulink
- Automated Tuning
- The systune reference page

H_2 performance, stability margin, pole location, and disturbance rejection requirements

New TuningGoal requirement objects allow you to specify a variety of tuning objectives for automated tuning of fixed-structure control systems with systune and looptune. New tuning requirements include:

- TuningGoal.Margins — Tune to stability margin requirements by specifying minimum gain and phase margins for any feedback loop in your control system.
- TuningGoal.Poles — Constrain closed-loop dynamics of your control system.
- TuningGoal.StableController — Constrain dynamics or ensure stability of tunable elements.

- `TuningGoal.WeightedGain` — Limit on frequency-weighted gain from specified inputs to specified outputs in your control system.
- `TuningGoal.Variance` and `TuningGoal.WeightedVariance` — Tune to H_2 performance requirements by minimizing or constraining variance amplification. `TuningGoal.Variance` specifies the maximum output variance for a unit-variance input signal from a specified input to a specified output in your control system. `TuningGoal.WeightedVariance` imposes a frequency-weighted variance amplification limit.

For more information about these `TuningGoal` requirement objects see the reference pages for each requirement object, and:

- Using Design Requirement Objects
- Specifying Design Requirements for systune
- Performance and Robustness Specifications for looptune

Robust tuning of one controller against a set of plant models

The new `systune` command can simultaneously tune the parameters of multiple models or control configurations. This feature allows you, for example, to tune a single controller against a range of plant models, to help ensure that the tuned control system is robust against parameter variations. As another example, you can tune for reliable control by simultaneously to multiple plant configurations that represent different failure modes of a system. In either case, `systune` finds values for tunable parameters that best satisfy the specified tuning objectives for all models.

For more information, see Tune Controller Against Set of Plant Models.

Option to constrain tuned parameter values and to restrict some tuning requirements to a frequency band

You can now optionally impose lower and upper bounds on tunable parameters when tuning fixed-structure control systems using `systune`, `looptune`, or `hinfstruct`. For example, you can constrain a gain to always be positive, or impose a maximum value on a filter time constant.

To impose bounds on tunable parameters, set the `Maximum` and `Minimum` properties of the parameter in the corresponding Control Design Block. For example, create a scalar gain block and constrain the gain to be positive:

```
gainblock = ltiblock.gain('gainblock',1,1);
gainblock.Gain.Minimum = 0;
```

Then, use `gainblock` as a component in a tunable `genss` model of the control system. When you tune the control system, the tuning command enforces the constraint.

Additionally, you can limit the range of frequencies in which almost any `TuningGoal` requirement is enforced for fixed-structure control system tuning with `systune` or `looptune`. The only exceptions are `TuningGoal.Variance` and `TuningGoal.WeightedVariance`.

For example, you can enforce a stability margin requirement in a frequency band extending for one decade on each side of the target gain crossover frequency.

To limit the range of frequencies in which a requirement is enforced, use the `Focus` property of the `TuningGoal` requirement object. For example, create a requirement that limits the gain from an input `du` to an output `u` to 10. Limit enforcement of the requirement to the frequency range 10–1000 rad/s.

```
Req = TuningGoal.Gain('du','u',10);
Req.Focus = [10 1000];
```

`ltiblock.pid2` and `loopswitch` objects for tuning two-degree-of-freedom PID controllers and marking loop opening sites for open-loop requirements

New Control Design Blocks in Control System Toolbox™ allow you to specify more control structures and more types of constraints for fixed-structure control system tuning in MATLAB:

- `ltiblock.pid2` — Tunable two-degree-of-freedom PID controller

- loopswitch — Control Design Block for specifying feedback loop opening locations in a tunable genss model of a control system

For more information, see the ltiblock.pid2 and loopswitch reference pages.

TuningGoal.MaxGain and GainLimit property renamed

Compatibility Considerations: Yes

The tuning requirement TuningGoal.MaxGain is now called TuningGoal.Gain. Additionally, the GainLimit property of that tuning requirement is now called MaxGain.

For more information, see the TuningGoal.Gain reference page.

Compatibility Considerations

Replace instances of TuningGoal.MaxGain in your code with TuningGoal.Gain. Replace references to the GainLimit property with MaxGain.

Options in hinfstructOptions and looptuneOptions renamed or removed

Compatibility Considerations: Yes

The following options in hinfstructOptions and looptuneOptions are changed:

- SpecRadius is now called MaxFrequency. Additionally, NaN is no longer a supported value for this option. For an unconstrained MaxFrequency value, use Inf.
- StableOffset is now called MinDecay.
- StableRadius option has no effect.
- StableExclude option of hinfstructOptions has no effect. hinfstruct now automatically excludes from stability tests Control Design Blocks

such as weighting functions or multipliers. These blocks do not affect the closed-loop stability of the actual control system to tune.

For more information about these options, see the `hinfstructOptions` and `looptuneOptions` reference pages.

Compatibility Considerations

If you use any of the affected options in your code, update your code to reflect the current names and supported values.

R2012a

Version: 4.1

New Features: Yes

Bug Fixes: Yes

Parallel Computing Support for looptune and hinfstruct

If you have Parallel Computing Toolbox™ software installed, you can use parallel computing to speed up tuning of fixed-structure control systems with the `looptune` or `hinfstruct` commands. When you run multiple randomized `looptune` or `hinfstruct` optimization starts, parallel computing speeds up tuning by distributing the optimization runs among MATLAB workers.

For more information about using parallel computing to speed up `looptune` or `hinfstruct` tuning, see:

- Speed Up Tuning with Parallel Computing Toolbox Software in the Robust Control Toolbox™ documentation.
- The Robust Control Toolbox demo Using Parallel Computing to Accelerate the Tuning Process.

For more information about tuning fixed-structure control systems with `looptune` or `hinfstruct`, see Tuning Fixed Control Architectures in the Robust Control Toolbox documentation.

Faster and More Accurate H-infinity Norm Computation Using SLICOT Algorithms

H_∞ norm calculations now use the SLICOT library of numerical algorithms. These algorithms improve the speed and accuracy of functions such as `hinfstruct` and `looptune`.

For more information about the SLICOT library, see http://slicot.org.

R2011b

Version: 4.0

New Features: Yes

Bug Fixes: Yes

looptune Tunes Fixed-Structure Control Systems

Use `looptune` to tune fixed-structure control systems to meet your requirements. To use `looptune`, specify design requirements such as loop bandwidth, stability margin, setpoint tracking, or target loop shape. `looptune` automatically tunes the parameters of your controller to meet the specified requirements.

The requirements objects `TuningGoal.MaxGain`, `TuningGoal.Tracking`, and `TuningGoal.LoopShape` let you express your design requirements directly. You do not have to first convert them to weighting functions or mathematical constraints on an optimization problem.

You can use `loopview` to validate the performance the performance of the tuned control structure against your specified design requirements.

For more information, see Tuning Fixed Control Architectures and the `looptune` and `loopview` reference pages.

Control System Tuning for Simulink Models with looptune or hinfstruct Using slTunable Interface

If you have Simulink Control Design software, you can use tuning commands, such as `slTunable.looptune` and `hinfstruct`, to tune control systems modeled in Simulink. The `slTunable` object provides an interface between your Simulink model and these commands.

Use `slTunable` to specify information about your control structure and parametrization. `slTunable` also automates tasks such as linearizing the Simulink model, parametrizing the tunable blocks of your system, and applying tuned parameter values to the model. After you create and configure an `slTunable` object for your control architecture, you can tune the control system using `slTunable.looptune` or `hinfstruct`.

For more information, see Tuning Fixed Control Architectures and the following demos:

- Tuning of a Digital Motion Control System

- Decoupling Controller for a Distillation Column
- Tuning of a Two-Loop Autopilot
- Tuning of Cascaded PID Loops
- Loop Shaping Design with HINFSTRUCT
- Fixed-Structure Autopilot for a Passenger Jet

wcgainplot for Visualizing Worst-Case Gains

`wcgainplot` plots the nominal, sampled, and worst-case gains of uncertain systems as a function of frequency. Use `wcgainplot` for visual analysis of uncertain systems.

For more information, see the `wcgainplot` reference page.

Functionality Being Removed or Changed

Compatibility Considerations: Yes

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`umat` object can no longer contain `ultidyn` or `udyn` uncertainty.	• Presence of `ultidyn` or `udyn` uncertain elements forces model type to `uss` or `ufrd` rather than `umat`. • Mixing `ureal` or `ucomplex` models with `udyn` or `ultidyn` objects	Expect a model type of `uss` or `ufrd` instead of `umat` when working with `udyn` or `ultidyn` uncertain elements.	Update code to work with `uss` or `ufrd` instead of `umat` when `udyn` or `ultidyn` elements are present.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
	produces uss instead of umat.		
uss(sys_frd), where sys_frd is a frd model object no longer converts sys_frd to ufrd.	Errors.	ufrd(sys_frd).	Replace uss(sys_frd) with ufrd(sys_frd).
ufrd(udat,freq,...) no longer constructs an uncertain frd model from the umat object udat.	Converts udat to a ufrd object with frequencies freq.	Use frd(udat,freq,...) to construct an uncertain frd model from the umat object udat.	Replace ufrd(udat,freq,...) with frd(udat,freq,...).
frd(sys_uss,w) where sys_uss is a uss model.	Warns; returns frd model containing data based on nominal response of sys_uss.	ufrd(sys_uss,w) to obtain a ufrd model.	Replace frd(sys_uss,w) with ufrd(sys_uss,w).
Nominal value of ultidyn object.	Nominal value is ss model object.	None.	Update code to work with ss model objects when working nominal value of ultidyn.
usubs.	Applied to array of uncertain models, default substitution is '-once'.	Use '-batch' to perform batch substitution on uncertain model arrays.	Replace usubs(...) with usubs(...,'-batch').
	usubs(M,{a1;a2;...},{v1;v2;...}) returns error.	usubs(M,a1,v1,a2,v2,...).	Replace usubs(M,{a1;a2;...},{v1;v2;...}) with usubs(M,a1,v1,a2,v2,...).

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`usample(sys,'a',na,'b',nb)` where uncertain element `b` does not exist in `sys`.	Returns `na`-by-`nb` array with constant values across `nb` dimension, instead of `na`-by-1 array.	None.	Update code to reflect correct dimensionality.
`wcgopt`.	Still runs.	`wcgainOptions` or `wcmarginOptions`.	Replace `wcgopt` with `wcgainOptions` or `wcmarginOptions`.
`robuststab` and `robustperf`.	For `ufrd` models, `BadUncertainValues` field of `Info` output returns `Nf`-by-1 struct array, where `Nf` is the number of frequency points.	None.	Update code to work with `Nf`-by-1 struct array for `BadUncertainValues` instead of `Nf`-by-1 cell array.
	For nominally unstable models, performance margin is zero (instead of a negative value).	None.	Update code to reflect correct performance margin .
`robopt`.	Still runs.	`robuststabOptions` or `robustperfOptions`.	Replace `robopt` with `robuststabOptions` or `robustperfOptions`.
`actual2normalized`.	First output argument is normalized uncertain block value. The second output argument is normalized distance between block value and nominal value.	`[NV,ndist] = actual2normalized(BLK,AV)`.	Use second output argument `ndist` for normalized distance.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
reshape(unc_sys,S).	S does not include the I/O size of the models in the array unc_sys. For example, if unc_sys is a 6-by-1 array of 2-output, 4-input models, reshape(unc_sys,[2 3]) converts unc_sys to a 2-by-3 array.	None.	Remove I/O size dimensions from reshape on uncertain model arrays.
diag(uss_sys) where uss_sys is a uss model.	Errors.	None.	Remove diag(uss_sys).

R2011a

Version: 3.6

New Features: Yes

Bug Fixes: Yes

Enhanced Workflow for H-Infinity Synthesis of Fixed-Structure Control Systems

New Generalized LTI models in Control System Toolbox allow you to model control systems with tunable parameters. Using these models simplifies controller tuning with `hinfstruct`. You can model a closed-loop transfer function, including tunable parameters, as a generalized state-space (`genss`) model and directly tune the parameters to minimize the closed-loop gain. The `hinfstruct` command can tune any fixed-structure SISO or MIMO control system using H_∞ synthesis techniques.

Additionally, new `realp` and `genmat` objects let you create parametric expressions. You can use such expressions to create custom tunable components. For example, you can define a low-pass filter parametrized by its cutoff frequency, or an observer-based controller parametrized by the state-feedback and observer gains.

For more information about creating tunable Generalized LTI models, see Models with Tunable Coefficients in the *Control System Toolbox User's Guide*.

For more information about H_∞ tuning with `hinfstruct`, see Tuning Fixed Control Architectures in the *Robust Control Toolbox Getting Started Guide*.

For examples of designing controllers for several different architectures using `hinfstruct`, see the following updated and new demos:

- Loop Shaping Design with HINFSTRUCT (updated)
- Tuning of a Two-Loop Autopilot (updated)
- Decoupling Controller for a Distillation Column (updated)
- Multi-Loop PID Control of a Robot Arm (updated)
- Fixed-Structure Autopilot for a Passenger Jet (new)

R2010b

Version: 3.5

New Features: Yes

Bug Fixes: Yes

New Commands for H-Infinity Synthesis of Fixed-Structure Control Systems

New commands in this release allow you to tune fixed-structure SISO and MIMO control systems using the techniques of H_∞ synthesis.

The new `hinfstruct` command lets you use the frequency-domain methods of H_∞ synthesis to tune control systems with a broad range of architectures and controller structures. For example, you can tune:

- Fixed-order, fixed-structure controllers, such as pure gains, PID controllers, or fixed-order transfer function or state-space models
- Single feedback-loop architectures with multiple tunable elements, such as a PID controller plus a filter
- Multiple feedback-loop architectures with multiple tunable elements

Specify the tunable elements of your system using the new parametrized Control Design blocks `ltiblock.gain`, `ltiblock.pid`, `ltiblock.tf`, and `ltiblock.ss`.

For examples of designing controllers for several different architectures using `hinfstruct`, see the following new demos:

- Loop Shaping Design with HINFSTRUCT
- Tuning of a Fixed-Structure Autopilot
- Decoupling Controller for a Distillation Column
- Multi-Loop PID Control of a Robot Arm

For more information, see Tuning Fixed Control Architectures in the *Robust Control Toolbox Getting Started Guide*.

R2010a

Version: 3.4.1

New Features: No

Bug Fixes: Yes

R2009b

Version: 3.4

New Features: Yes

Bug Fixes: Yes

New Option to Improve Robust Performance by Accounting for Real Uncertain Parameters

You can now improve robust performance by accounting for real uncertain parameters when designing controllers using μ-synthesis. The user-defined options you use in the `dksyn` command now includes a new option `MixedMU`. Set this option to `'on'` to account for real uncertain parameters in your system. For more information, see the `dkitopt`, and `dksyn` reference pages.

New Command to Linearize Simulink Models with Uncertainty

If you have Simulink Control Design software installed, you can take model uncertainty into account when linearizing a Simulink model. You can then use the resulting uncertain linearized model (`uss` object) to perform linear analysis and robust control design.

If your model already contains Uncertain State Space blocks, use the new `ulinearize` command to obtain an `uss` model. If you want to account for uncertainty in your linear analysis without using Uncertain State Space blocks, you can specify individual Simulink blocks to linearize to an uncertain variable. For more information, see "Computing Uncertain State-Space Models from Simulink Models" in the *Robust Control Toolbox User's Guide*.

New Interface for Simulating Effects of Uncertainty in Simulink Models

This version of the product provides a new interface to simulate the effects of uncertainty in Simulink models. The interface includes the following:

- Uncertain State Space block to specify uncertain system in Simulink. You should replace USS System blocks in your existing models with the Uncertain State Space block. To do so, run the `slupdate` command on your models.
- `ufind` command to extract all uncertain variables from a Simulink model.
- `usample` command to generate random values of these uncertain variables.

For more information on simulating the effects of uncertainty using the new interface, see "Simulating Effects of Uncertainty" in the *Robust Control Toolbox User's Guide*.

New Command to Model Multiple LTI Responses as One Uncertain System

This version of the product includes a new `ucover` command that lets you model a family of LTI responses as one uncertain system. For more information, see the `ucover` reference page.

New and Updated Demos

The following new and updated demos illustrate use of the new features:

- "Control of Spring-Mass-Damper Using Mixed mu-Synthesis" shows use of the new `MixedMU` option and `dksyn` command for mixed-mu synthesis.
- "Linearization of Simulink Models with Uncertainty" shows how to compute uncertain state-space models using `ulinearize` and Simulink Control Design software.
- "Robustness Analysis in Simulink" uses the new interface for simulating effects of uncertainty in Simulink models.
- "Simultaneous Stabilization Using Robust Control" and "Modeling a Family of Responses as an Uncertain System" show use of the `ucover` command.
- "First-Cut Robust Design" shows use of the `usample`, `ucover` and `dksyn` commands.

To access the demos, type

```
demo('toolbox','robust control')
```

Functions, Properties and Blocks Being Removed

Compatibility Considerations: Yes

Function, Property or Block Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
`usiminfo`	Still runs	`ufind`	See "New Interface for Simulating Effects of Uncertainty in Simulink Models" on page 40.
`usimfill`	Still runs	`ufind`	See "New Interface for Simulating Effects of Uncertainty in Simulink Models" on page 40.
`usimsamp`	Still runs	`usample`	See "New Interface for Simulating Effects of Uncertainty in Simulink Models" on page 40.
USS System block	Still runs	Uncertain State Space block	See "New Interface for Simulating Effects of Uncertainty in Simulink Models" on page 40.
`ltiarray2uss`	Still runs	ucover	See "New Command to Model Multiple LTI Responses as One Uncertain System" on page 41.

R2009a

Version: 3.3.3

New Features: No

Bug Fixes: Yes

R2008b

Version: 3.3.2

New Features: No

Bug Fixes: Yes

R2008a

Version: 3.3.1

New Features: Yes

Bug Fixes: No

Ability to Use LOOPMARGIN with Simulink

This version of Robust Control Toolbox software lets you analyze the robustness of nonlinear Simulink models using the LOOPMARGIN command.

If you have the Simulink Control Design product installed, you can perform stability margin analysis of a Simulink model by passing the model name and a point within that model to the LOOPMARGIN command.

R2007b

Version: 3.3

New Features: No

Bug Fixes: No

No New Features or Changes

R2007a

Version: 3.2

New Features: Yes

Bug Fixes: No

New Simulink Blocks

- USS System — This Robust Control Toolbox version introduces a new Simulink block, USS System. You can use this block to import uncertain systems into Simulink models.
- Multiplot Graph — Plot multiple signals in one figure.

R2006b

Version: 3.1.1

New Features: Yes

Bug Fixes: No

New Function ltiarray2uss

This Robust Control Toolbox version introduces a new function, `ltiarray2uss`. This function constructs an uncertain state-space model from an LTI array.

R2006a

Version: 3.1

New Features: No

Bug Fixes: No

No New Features or Changes

R14SP3

Version: 3.0.2

New Features: No

Bug Fixes: No

No New Features or Changes

R14SP2

Version: 3.0.1

New Features: Yes

Bug Fixes: No

mussvunwrap Is Renamed

`mussvunwrap` has been renamed. It is now called `mussvextract`.

New Functions actual2normalized and normalized2actual

This Robust Control Toolbox version introduced two new functions:

- `actual2normalized` — Calculate normalized distance between nominal value and given value for uncertain atom.
- `normalized2actual` — Convert value for atom in normalized coordinates to corresponding actual value.